An Event Modeling and Analysis Method Based on Bipartite Graph

Mei Xu [a], Yanlei Shang [b]

Institute of Network Technology Research Center, Beijing University of Posts and Telecommunications, Beijing 100876, China

[a]xumeigogo@163.com, [b]shangyl@bupt.edu.cn

Abstract. As information systems become larger and more complex, maintenance and failure recovery become more difficult. Usually, these information systems monitor system state and business logic by logs or events. Analyzing these events which contain vital information in-depth is very necessary for both business intelligence, trouble shooting and event mining. There are several excellent event analysis methods including Complex Event Processing (CEP) which focus on processing complex events composed of single events. However, CEP cannot compute the relationship between events easily. In this paper, we propose a method for modeling events using Bipartite Graph. Besides, we achieve a vertex-centric algorithm that can be executed parallel to analyze the relationship between events or event sources. After that, we prove the feasibility and validity of the modeling method and algorithm with a sample experiment.

Keywords: Event Modeling; Vertex-centric Algorithm; Bipartitie Graph; Event Analysis.

1. Introduction

Event–based systems, applications and event-driven architecture achieve increasing popularity in the recent years. Besides, with the development of Internet of Things (IoT) tremendous number of sensors were deployed [1]. Also, more and more websites hope to use business intelligence to provide their customers the information they look for accurately. The systems, sensors and website users are producing events at all time. These events contain important and interesting information which records the systems run time situation, the temperature and pressure data collected by sensors, the users' behavior generated by keystrokes and mouse click etc.

There are generally two approaches for event analysis, CEP and log analysis. However, neither of the two methods could be used easily and efficiently to analysis batch events deeply. Recent years, graph analysis and graph database have become gradually popular in Business Intelligence and Social Relationship Analytics. Graph can model different objects and their relationships in a natural and intuitive way. Bipartite Graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V[3]. Many data types can be modeled as Bipartite Graphs, for instance, reviewers and movies in movie recommender system, customers and pursing items in market basket analysis [4].

In this paper, we propose a method for modeling events into a Bipartite Graph composed of two sets of vertexes respectively present event source and event. We achieve a vertex-centric algorithm to compute the similarity between events in same type. After that, we design and implement an event storage and analysis system and prove the usefulness of the system with a sample experiment.

The paper is structured as follows: Section II describes related work. Section III gives a high-level architecture of our event storage and analysis system. In Section IV, we introduce the implementation of the event storage and analysis system. Section V shows the sample experiment and the resultant graph which uses the e-business user's events as input. Finally, we make a conclusion in Section VI.

2. Related Work

CEP and log analysis are the two common methods. CEP is used to identify meaningful events through multiple event streams in real-time. A number of CEP engines were arose: Stream Base enables rapid development of real-time analytical applications with pre-built integration for streaming and historical data [7]; EsperTech provides Event Processing Language (EPL) designed for concisely expressing situations and fast execution against both historical and currently-arriving events [8];

Microsoft Stream Insight, a part of SQL Server 2008 R2, supports application to query processing for events within a time window [9].

The log analysis tools like Webalizer, Awstats, and Google Analytics are either standalone or have the limitation of data scale [10]. Others like Splunk can handle any kind of text log file and it is appropriate for analyzing line logs. Also, there still some researchers are devoting themselves to study how to provide the generality and efficiency of log analysis, James Carey et al. [6] provided a toolkit for event analysis named TEAL to analyze events. Someone think only a subset of the events in log requires attention, Theodoros Kalamatianos et al. [2] only chose some beacon event. Some others are researching how to use cloud computing to do batch processing, [10] presents a unified cloud platform for batch log analysis with the combination of Hadoop and Spark.

As CEP focus on real-time processing and log analysis needs to compare a large volume of logs. The method we proposed in this paper has more advantages, Bipartite Graph can present events naturally, and the graph database can be used to store the event graph. Also, with the graph analysis and Titan's [5] On-Line Analytical Processing (OLAP) interface, the relationship between events and event sources can be compute in a big data platform.

3. Architecture of Our Event Modeling and Analysis System

In this section, we introduce the architecture of event storage and analysis system which is shown in Fig. 1. We also present our event modeling method and vertex-centric algorithm respectively.

Our system composes of three layers, Event Pre-Processing Layer (EPPL) is responsible for events collection and information filtering, Event Storage Layer (ESL) models events into graph factors and persists it into graph database, and if complex analysis is required the Event Analysis Layer (EAL) would be worked.

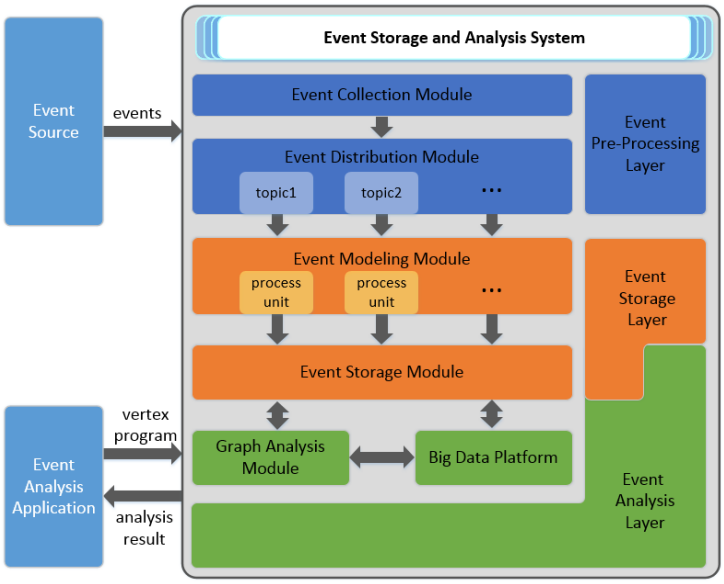


Fig. 1 The architecture of the event storage and analysis system

EPPL consists of two modules, Event Collection Module (ECM) and Event Distribution Module (EDM). ECM listens to external event source all the time. When an event produced ECM pulls the event and filters out the fields won't be analyzed. EDM is responsible for distributing the events to Event Storage Layer.

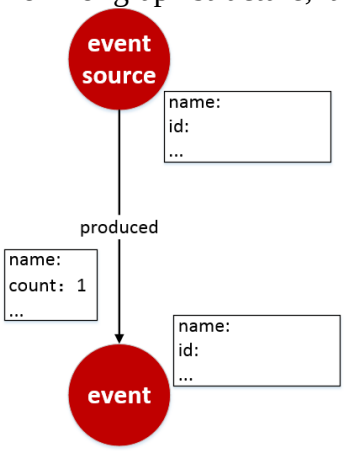
ESL includes Event Modeling Module (EMM) and Event Storage Module (ESM). Process unit models the event into vertexes and an edge when event arrivals and writes these graph factors into ESM. ESM is responsible for graph data storage and transmitting events with EAL.

Graph Analysis Module (GAM) and Big Data Platform (BDP) constitute EAL which is responsible for large scale, complex analysis. GAM receives vertex program to do complex analysis on the event graph. Then the vertex program is copied to BDP to execute. GAM also does the graph

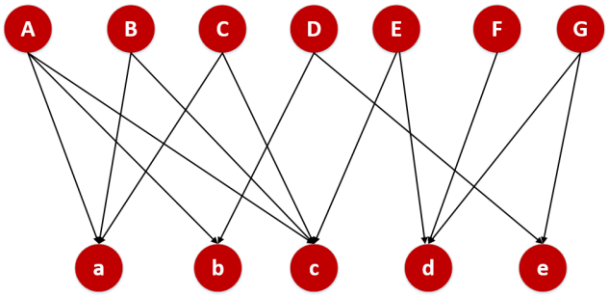
merging with a period of time. BDP provides the complex analysis an environment which can be Spark or Hadoop to execute user's vertex program in parallel.

3.1 Event Modeling Method

In this subsection, we describe our modeling method based on Bipartite Graph. As a common data structure, graph is widely used in presenting complicated and general relations such as sensor networks. Bipartite Graph is a special form of graph structure, it can be used to present a lot of scenes.



(a) Graph structure of a single event



(b) Graph structure of multiple events

Fig. 2 The event source-event bipartite graph

Table 1 Required properties of event source-event bipartite graph

Graph Factor	Property Name	Description
Event Source Vertex	id	Unique identifier of event source, provided by graph database.
	name	Identifier of event source which presents the event source intuitive.
	label	Identify the type of vertex, default value is 'event source'.
Event Vertex	id	Unique identifier of the event, usually provided by graph database.
	name	Identifier of the event which can presents event intuitive.
	label	Identify the type of vertex, default value is 'event'.
Edge	id	Unique identifier of edge, usually provided by the graph database.
	label	Identify relationship of two vertexes, default value is 'produced'.
	count	The number of event source produced event, default value is 1.

We propose a method for modeling event based on Bipartite Graph. We divide an event into two objects, the event source that the event producer and the event itself. The event source-event bipartite graph can be described as Figure 2. A graph is usually defined as$\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ presents vertexes and $\mathbf{E}$ presents the edges. We defined the event source-event bipartite graph as follows:

$$\mathbf{G_{bg}} = (\mathbf{X}, \mathbf{E}, \mathbf{Y}), \mathbf{X} \cup \mathbf{Y} = \mathbf{V}, \mathbf{X} \cap \mathbf{Y} = \emptyset, \mathbf{E_{ij}} \in \mathbf{E}(\mathbf{i} \in \mathbf{X}, \mathbf{j} \in \mathbf{Y})$$

$\mathbf{G_{bg}}$ is the event source-event bipartite graph, **X** and **Y** constitute the vertex set of the graph and **X** and **Y** are disjoint. $\mathbf{E_{ij}}$ expresses edge which source vertex is **i** and destination vertex is **j**. As Figure 2(a) shown, a single event can be modeled as two vertexes and one edge, vertex and edge have required properties and user-defined properties, the properties is shown in Table 1. As the events amount increasing, a huge and standard bipartite graph can be produced shown as Figure 2(b).

3.2 The vertex-centric algorithm

SimRank is a powerful model for assessing vertex-pair similarities in a graph [11]. Inspired by it, we propose a vertex-centric algorithm to compute vertex-pair in event source-event bipartite graph.

In bipartite graphG_{bg}, the similarity between vertex A and vertex B depends on the probability that the two vertexes meeting based on random walking. Let $\mathrm{P(A,a)}$ denotes the probability that vertex A walk to vertex a, and $\mathrm{s(A,B)}$ denotes the similarity between the two objects. The method can be obtained as follows:

$$\mathrm{P(A,a)} = \frac{count(A,a)}{\sum_{a\in O(A)} count(A,a)} \tag{1}$$

$$\mathrm{s(A,B)} = \sum_{c\in(O(A)\cap O(B))} \frac{P(A,c)\times P(B,c)}{|I(\mathrm{c})|} \tag{2}$$

Where $\mathrm{count(A,a)}$ is the count property of the edge from A to a which presents the number of edges from event source indicate event. $\mathrm{O(A)}$ is the set of out-neighbors of A, $I(\mathrm{c})$ is the set of in-neighbors of c, $|I(\mathrm{c})|$ is the number of elements in set $I(\mathrm{c})$.

4. Implementation of Our Event Modeling and Analysis System

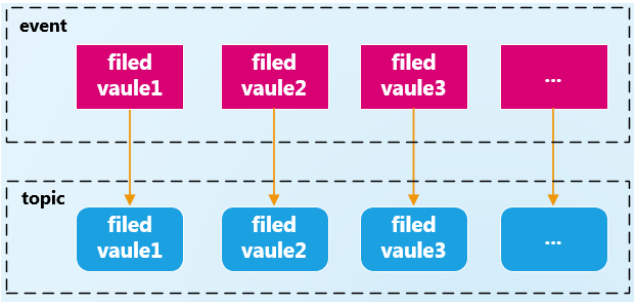
According the description of the event storage and analysis system in the above section, we will introduce some key technologies in this part.

4.1 Event Pre-Processing Layer

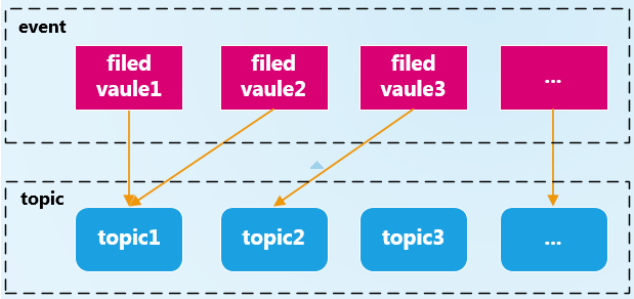
The events to be analyzed may come from a variety of sources like systems, sensors or user click. And the events may contain some information that analysis is not required. The events collected and filtered by ECM are send to EDM to publish to corresponding topics.

(1) Event Collection Module

User need to give the event format and fields should be analyzed. ECM filters out the unnecessary information according to the user's definition and then sends the event to next module.



(a) Mapping of event to topic in infinite number mode



(b) Mapping of event to topic in fixed number mode

Fig. 3 The mapping of event to topic

(2) Event Distribution Module

This module is responsible for the distribution of multiple events. In practice, we use Apache ActiveMQ and publish-subscribe mode to complete the distribution. User can choose topic's number according to the speed of their events produced. We provide two modes, infinite number mode and fixed number mode. Figure 3 shows the mapping of event to topic in the two modes.

In both modes, user should specify a field of the event which EDM creates topic according to. As Figure 3(a), in infinite number mode, the mapping of event to topic is one-to-one, EDM will create a new topic when a new type of event filed came and publish the event to corresponding topic. In fixed number mode shown in Figure 3(b), the mapping of event to topic is many-to-one, EDM creates a fixed number of topics which specified by user. In this mode, EDM needs to compute the corresponding topic index, the computing method is described as follows:

1) Extracting the filed from the event;
2) Computing the hashcode of the filed;
3) Getting the topic index through $(\text{hashcode} + 1)\%(\text{topicNumber})$.

After getting the topic index, EDM sends the event to corresponding topic and process next event.

4.2 Event Storage Layer

ESL subscribes events from EDM, and then models the event to an event source-event bipartite graph structure by EMM, and last persists the bipartite graph to ESM.

(1) Event Modeling Module

EMM is designed to do event modeling and persisting. It maintains a *topicSet* and modeling schema. The items in the topicSet is topic id and topic url which created by EDM.

Different from topics in EDM, the number of the process unit usually fixed. One process unit subscribes one or more topics in EDM. In order to subscribe proper topics, the process unit should know what the topics are and how to connect to them. When one new topic is created by EDM, the topic should be registered to EMM and subscribed by one process unit, the process is as follows:

1) EDM creates a new topic and send id and url to EMM;
2) EMM receives the topic creating message and put the id and url into *topicSet*;
3) EMM computes the process unit index by formula (3)

$$(\text{hashcode}(\text{topicId}) + 1)\%(\text{processUnitNumber}) \tag{3}$$

4) Corresponding process unit subscribes the new topic.

The modeling schema is the base of the event modeling, the information of the schema includes: name and attributes of event source vertex; name and attributes of event vertex; attributes of edges.

With *topicSet* and the modeling schema, process unit receives events from topics it subscribed and models the events to graph structure based on the modeling method we described in Section III, The specific working flow of the process unit is as follows:

1) Making event source vertex and event vertex according to the schema;
2) Making an edge by connecting event source vertex to event vertex;
3) Writing the event to Event Storage Module;
4) Processing the next event.

(2) Event Storage Module

ESM is used to store events and results of analysis. In practice, we use Titan as ESM. It provides the way for our system to write and read the graph in it.

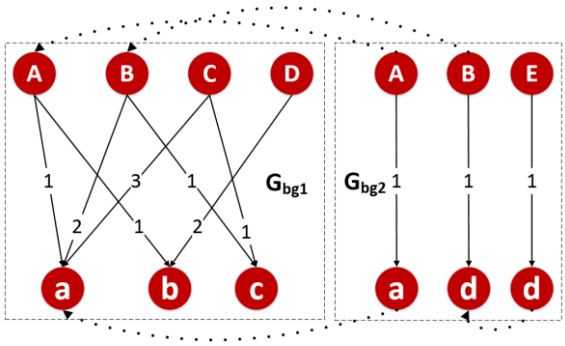
4.3 Event Storage Layer

EAL is the main part to do event analysis. It provides On-Line Transaction Process (OLTP) and OLAP. In our system, OLAP can be worked on Big Data Platform. To do analysis the user should provide the vertex program to EAL, the vertex program should be SQL-like fragment to do OLTP or a piece of vertex-centric code to do OLAP.

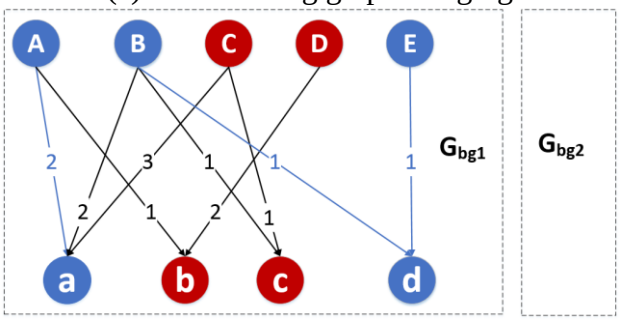
(1) Graph Analysis Module

GAM is responsible for three jobs: doing the graph merging at regular intervals; interacting with Event Analysis Application including receiving the vertex program and returning the analysis result; sending vertex program to BDP.

1) Graph Merging.



(a) Before doing graph merging



(b) After doing graph merging

Fig. 4 Graph merging process

In ESL, every event is stored as a single graph structure which is separated from the others. In fact, the two different original events may have the same event source or the same event itself. So the event source vertex and the event vertex of the two different events can be merged. The process of graph merging is shown in Figure 4. G_{bg1} in Figure 4(a) is the original graph and G_{bg2} is new events which need to be merged into G_{bg1}. Figure 4(b) shows the graph after merging, the vertexes and edges in color blue are the new factors. The graph merging algorithm is shown in Algorithm 1.

Algorithm 1: Graph Merging

```
// A as vertex array, save the vertex with label event source in Gbg2
A ← Gbg2.V().hasLabel('event source')
//Loop to merge every event in Gbg2 into Gbg1
//esv1 and ev1 respectively are event source vertex and event vertex in Gbg1; esv2 and ev2
respectively are event source vertex and event vertex in Gbg2
for i ← 0 to size (A)
  esv2 ← A[i]; // esv2 is the event source vertex in Gbg2
  esv2Name ← esv2.value( 'name')
  esv1 ← Gbg1.V( ).hasName('esv2Name')
  if ( esv1 is null)
     esv1← copy of (esv2)
  end if
 ev2 ← esv2.out( )
  ev2Name ← ev2.value ('name')
  ev1← Gbg1.V().hasLabel('event')
 .hasName( 'ev2Name')
  if (ev1)
     B ← esv1.out( )
     if (ev1 in B)
        count ( esv1 →ev1) + 1
     else
        make an edge from esv1 to ev1
     end if
```

```
      else
        ev1← copy of (ev2)
        make an edge from esv1 to ev1
      end if
    end for
drop Gbg2
```

2) Vertex Program

The vertex program is a piece of code to execute on BDP to do analysis. We archive a vertex-centric algorithm to compute the similarity of the vertexes with the same type in the event source–event bipartite graph. Inspired by SimRank, the algorithm contains the *count* attribute of the edge and the vertex-centric idea. The vertex-centric algorithm is shown in Algorithm 2.

Algorithm 2: Vertex-centric Algorithm

```
// as an example, we compute the similarity between the event source vertexes
// A as an vertex array, saves all the event source vertexes
A ← GbgV( ).hasLabel('event source')
for i ← 0 to size (A)
  evs1 ← A[i]
  for j ← 1 to size (A)
s ← 0 // the similarity between evs1 and evs2
    evs2 ← A[j]
//evs1_ov and evs2_ov respectively save their out neighbors
    evs1_ov ← evs1.out( )
    evs2_ov ← evs2.out( )
    ov ← evs1_ov ∩ evs2_ov
    for k ← 0 to size (ov)
      s ← s+ (P( evs1,ov[k]) * P(evs2, ov[k]) / in(ov))
    end for
    create edge between evs1 and evs2
    weight (edge) ← s
  end for
end for
```

(2) Big Data Platform

BDP can be integrated with Spark, Giraph or Hadoop to improve the speed. It provides environment to do complex analysis. GAM sends user-defined vertex program to it, then it copies the program to every worker and execute respectively. When the computation is accomplished, the result is persisted back to ESM or retuned to GAM to show to user.

5. Experiment and Evaluations

In this section, we will experiment with our event store and analysis system. We use the users' activity log in TianChi [12] as the event set, ActiveMQ as EDM, Titan as ESM, Spark's local mode as BDP. We model every event in this set, store them to Titan and compute the similarity between every event source vertex.

5.1 Modeling and Distribution

Each line in the event set is composed of user base information and this user's activity log. The user base information includes user_id, age_range, gender and merchant_id, the activity log contains the user's click events which are separated by #. Every event is composed of item_id, category_id, brand_id, time_stamp, action type.

As experiment, we filter out merchant_id in user base information, category_id, brand_id and timestamp in activity log. Also, we divide the activity log into multi click events and assemble each

event with the user base information to form a complete user event. Thus, every event passed through ECM can be described as user_id: age range: gender: item_id: timestamp: action type.

As the action type is divided into four types, we create four topics in ActiveMQ. Every topic is corresponding to one type events. Also we set up four process units in EMM and every process unit subscribe to one topic.

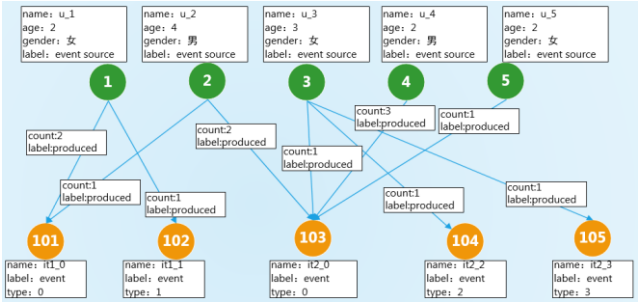


Fig. 5 The event source – event bipartite graph after modeling

In process unit, we model every user event to bipartite graph factor. We model the user as the event source vertex and the action as the event vertex. The schema of the user event is shown in Table 2. The event source–event bipartite graph after modeled is shown in Figure 5.

Table 2 Event source-event bipartite graph schema

Graph Factor	Property Name	Description
Event Source Vertex	id	Unique identifier of event source vertex, provided by Titan.
	name	A string, the value of filed user id, cannot be null.
	label	A string, event source, cannot be null.
	age	An integer, a range of ages, value of filed age range, can be null. 1: <18; 2:[18,24]; 3:[25,29]; 4: [30, 40]; 5:[40,50]; 6: >50.
	gender	A string, value of filed gender, can be null.
Event Vertex	id	The unique identifier of event vertex, provided by Titan.
	name	A string, the value of filed item id _ action type, such as 12_0, cannot be null.
	label	A string, event, cannot be null.
	type	An integer, the value of filed action type, cannot be null. 0: brows; 1: add to favorite; 2: add to chart; 3: purchase.
Edge	id	The unique identifier of the edge, provided by Titan.
	label	A string, produced, cannot be null.
	count	An integer, the number of the event source produced the event, the default value is 1, cannot be null.

5.2 Storage

In our experiment, we use Titan as graph database, use Cassandra as Titan's storage backend and Elasticsearch as it's index backend. We choose 10000 events from event set from TianChi and store them to ESM after pre-processing and modeling. The part of the graph is demonstrated in Figure 6.

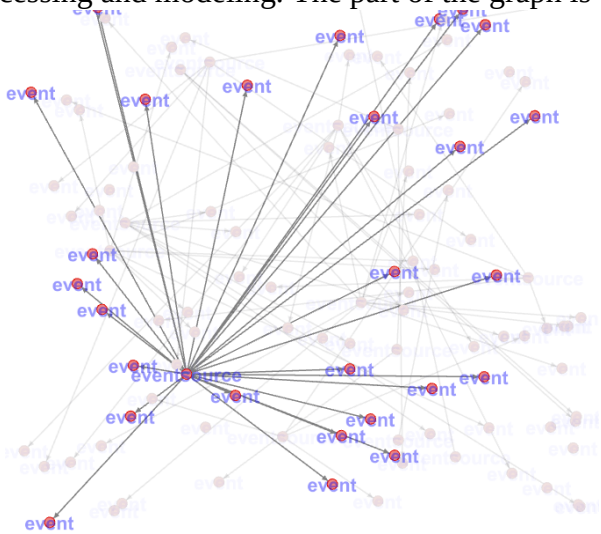


Fig. 6 A part of the event source–event bipartite graph

5.3 Event source – Event Bipartite Graph Analysis

We do OLTP and OLAP on the event source–event bipartite graph. The OLTP contains search all event source vertexes, lookup all event vertexes, and find out all events that one event source produced. The part of the result is shown in Figure 7.

```
gremlin> graph=TitanFactory.open('../conf/titan-cassandra.properties')
==>standardtitangraph[cassandrathrift:[10.108.167.215]]
gremlin> g=graph.traversal()
==>graphtraversalsource[standardtitangraph[cassandrathrift:[10.108.167.215]], standard]
gremlin> g.V().hasLabel('eventsource').valueMap()
==>[gender:[女], name:[177024], age:[4]]
==>[gender:[女], name:[167040], age:[5]]
==>[gender:[女], name:[375168], age:[3]]
==>[gender:[女], name:[110976], age:[5]]
==>[gender:[女], name:[48768], age:[5]]
==>[gender:[男], name:[234624], age:[0]]
==>[gender:[女], name:[34176], age:[6]]
==>[gender:[女], name:[240768], age:[2]]
==>[gender:[女], name:[181632], age:[3]]
==>[gender:[女], name:[302208], age:[2]]
```

(a) A part result of searching all event source vertexes

```
gremlin> g.V().hasLabel('event').valueMap()
==>[name:[253179_0], type:[0]]
==>[name:[202852_0], type:[0]]
==>[name:[842483_0], type:[0]]
==>[:]
==>[name:[358937_0], type:[0]]
==>[name:[677580_0], type:[0]]
==>[name:[557248_0], type:[0]]
==>[name:[755042_0], type:[0]]
==>[name:[359620_0], type:[0]]
==>[name:[655967_0], type:[0]]
```

(b) A part result of searching all event vertexes

```
gremlin> g.V(40964152).out().valueMap()
==>[name:[403553_0], type:[0]]
==>[name:[922914_0], type:[0]]
==>[name:[700267_0], type:[0]]
==>[name:[217780_0], type:[0]]
==>[name:[677580_0], type:[0]]
==>[name:[557248_0], type:[0]]
==>[name:[105451_0], type:[0]]
==>[name:[203177_0], type:[0]]
```

(c) A part result of finding one user's events

Fig. 7 Result of OLTP

We use our vertex-centric algorithm described in Section IV to compute the similarity between users. Figure 8 is the partial result.

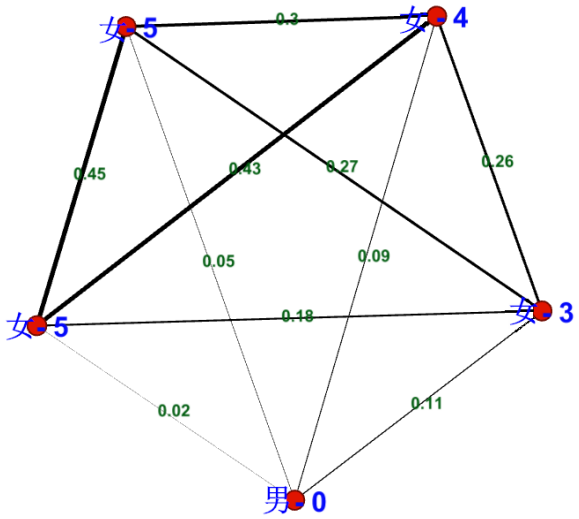


Fig. 8 Partial result of similarity computation

6. Conclusion And Future Work

In this paper, we contribute a noval event modeling method and a vertex-centric algorithm. We model events to an event source-event bipartite graph through dividing an event into two vertexes and one edge. Based on the event source-event bipartite graph, we propose a vertex-centric algorithm to compute the similarity between two vertexes with same type. This algorithm can execute parallel on big data platform such as Spark to improve speed. At last, we do a sample experiment to calculate similarity between event sources. We choose TianChi's user activity log as our event set and fetch 10000 events as input. We get the similarities of users by modeling these events using our method and analyze using our vertex-centric algorithm. The results of the experiment confirmed the feasibility and validity of our event modeling method and vertex-centric algorithm.

References

[1]. Ruben Mayer, Boris Koldehofe and Kurt Rothermel, "Predictable low latency event detection with parallel complex event processing," IEEE Internet of Things Journal, 2015, 2(4): 274-286.

[2]. Theodoros Kalamatianos, Kostas Kontogiannis and Peter Matthews, "Domain independent event analysis for log data reduction," 2012 IEEE 36th Annual Computer Software and Applications Conference, 2012, 225-232.

[3]. https://en.wikipedia.org/wiki/Bipartite_graph.

[4]. Hongyuan Zha, Xiaofeng He, Chris Ding, et al., "Bipartite graph partitioning and data clustering," Proceedings of the tenth international conference on Information and knowledge management[C], 2001:25-32.

[5]. http://titan.thinkaurelius.com/.

[6]. James Carey and Philip Sanders, "A toolkit for event analysisi and logging," 2011 International Conference for High Performacne Computing, Networking, Storage and Analysis (SC), 2011, 1-7.

[7]. http://www.streambase.com/.

[8]. http://www.espertech.com/.

[9]. https://msdn.microsoft.com/en-us/library/ee362541 (SQL.105).aspx.

[10]. Xiuqin Lin, Peng Wang, and Bin Wu, “Log analysis in cloud computing environment with Hadoop and Spark,” Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE Internatiional Conference, 2013: 273-276.

[11]. Weiren Yu, Xuemin Lin, Wenjie Zhang and Julie A. Mc Cann, “Fast all-pairs SimRank assessment on large graphs and bipartite domains,” IEEE Transactions on Knowledge and Data Enginerring, 2015, 27(7): 1810-1823.

[12]. https://tianchi.shuju.aliyun.com/.